Kauno technologijos universitetas

Informatikos fakultetas

Miglotoji logika

P176B101 Intelektikos pagrindai

Antras laboratorinis darbas

Autorius
Gustas Klevinskas

Akademinė grupė
IFF-8/7

Vadovai
Lekt. dr. Audrius Nečiūnas
Doc. dr. Agnė Paulauskaitė-Tarasevičienė

Kaunas, 2021

Turinys

Įvadas

Antrojo laboratorinio darbo tikslas – sumodeliuoti miglotosios logikos sistemą sugalvotai sričiai. Šiam tikslui įgyvendinti iškelti uždaviniai:

1. Sugalvoti miglotosios logikos sistemą.
2. Pasitikrinti sumodeliuotą sistemą naudojant Matlab fuzzy įrankį.
3. Realizuoti miglotosios logikos sistemą naudojant Python.

1. Užduoties aprašas

1.1. Parametrai

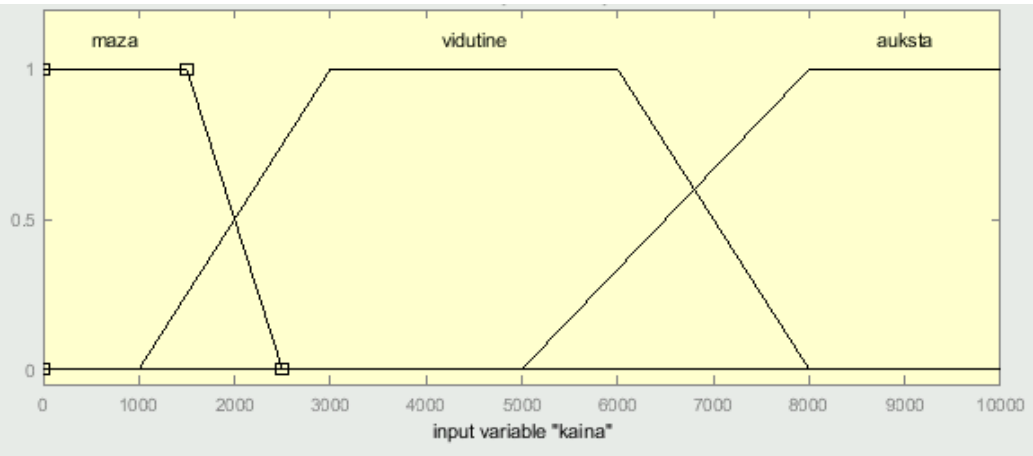
Pasirinkta realizuoti sistemą, kuri nustato automobilio pirkimo tikimybė atsižvelgiant į tris jo parametrus: kainą, ridą ir metus. Rezultato parametras – tikimybė, jog automobilis bus perkamas. Šie parametrai parinkti pagal asmeninius kriterijus, o ne gauti naudojant išorinius šaltinius.

1.1.1. Kaina

1 lentelėje pateiktos trapecijų viršūnės kainos įverčiui. Paveiksliuke žemiau pavaizduotos šio įverčio trapecijos.

1 lentelė. *Kainos fuzzy aibės reikšmės*

Kainos įvertis	Trapecijos viršūnės *x* ašyje
Maža	0, 0, 1500, 2500
Vidutinė	1000, 3000, 6000, 8000
Aukšta	5000, 8000, 10000, 10000



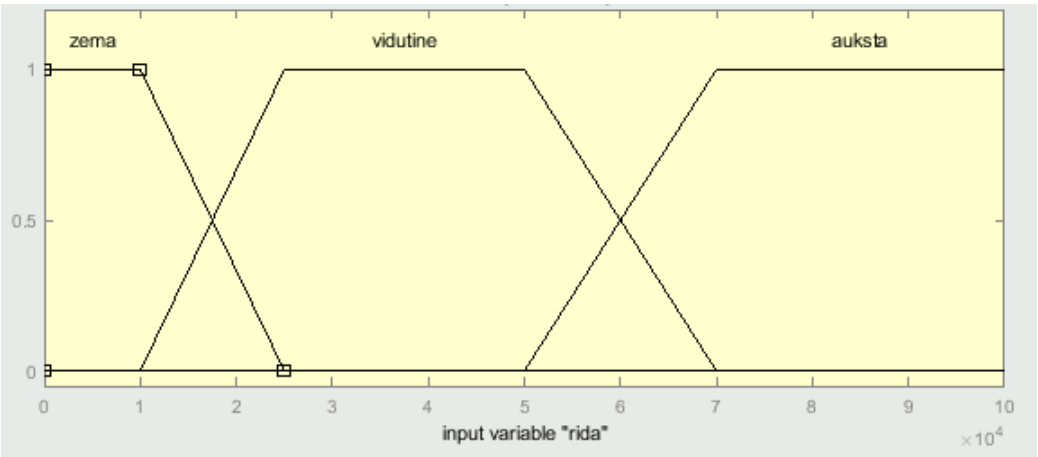
1 pav. *Kainos fuzzy aibė*

1.1.2. Rida

2 lentelėje pateiktos trapecijų viršūnės ridos įverčiui. Paveiksliuke žemiau pavaizduotos šio įverčio trapecijos.

2 lentelė. *Ridos fuzzy aibės reikšmės*

Ridos įvertis	Trapecijos viršūnės *x* ašyje
Žema	0, 0, 10000, 25000
Vidutinė	10000, 25000, 50000, 70000
Aukšta	50000, 70000, 100001, 100001



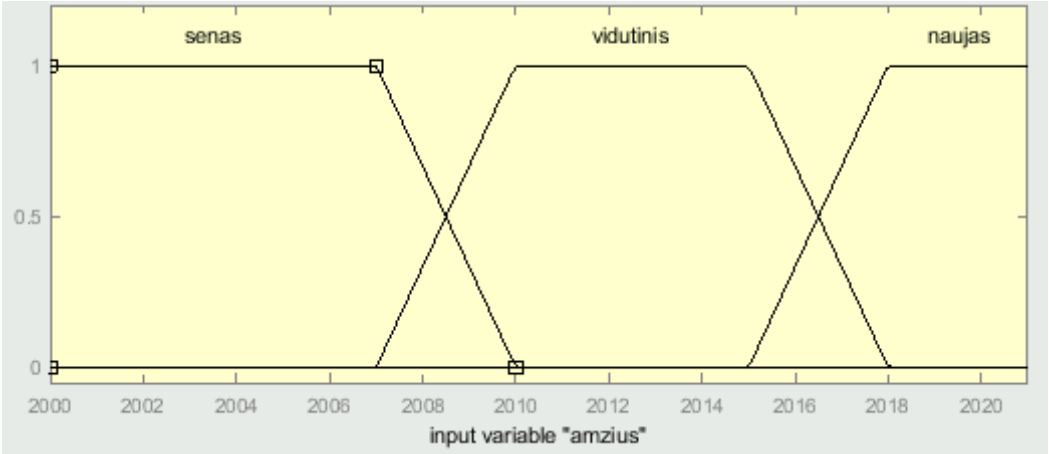
2 pav. *Ridos fuzzy aibė*

1.1.3. Amžius

3 lentelėje pateiktos trapecijų viršūnės amžiaus įverčiui. Paveiksliuke žemiau pavaizduotos šio įverčio trapecijos.

3 lentelė. *Amžiaus fuzzy aibės reikšmės*

Amžiaus įvertis	Trapecijos viršūnės *x* ašyje
Sena	2000, 2000, 2007, 2010
Vidutinė	2007, 2010, 2015, 2018
Nauja	2015, 2018, 2022, 2022



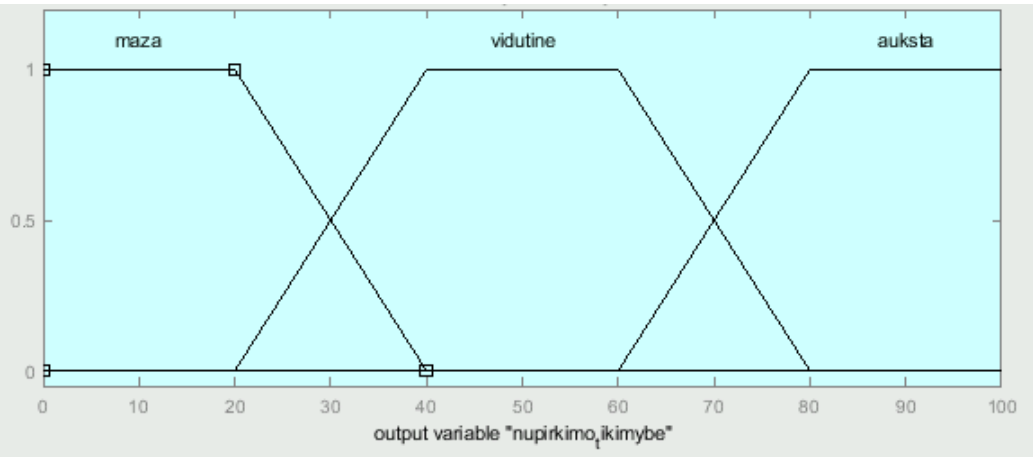
3 pav. *Amžiaus fuzzy aibė*

1.1.4. Nupirkimo tikimybė

4 lentelėje pateiktos rezultatinių trapecijų viršūnės nupirkimo tikimybės įverčiui. Paveiksliuke žemiau pavaizduotos šio įverčio trapecijos.

4 lentelė. *Nupirkimo tikimybės fuzzy aibės reikšmės*

Nupirkimo tikimybė	Trapecijos viršūnės *x* ašyje
Žema	0, 0, 20, 40
Vidutinė	20, 40, 60, 80
Aukšta	60, 80, 100, 100



4 pav. *Amžiaus fuzzy aibė*

1.2. Taisyklės

Sistemos taisyklės pateiktos 5 lentelėje.

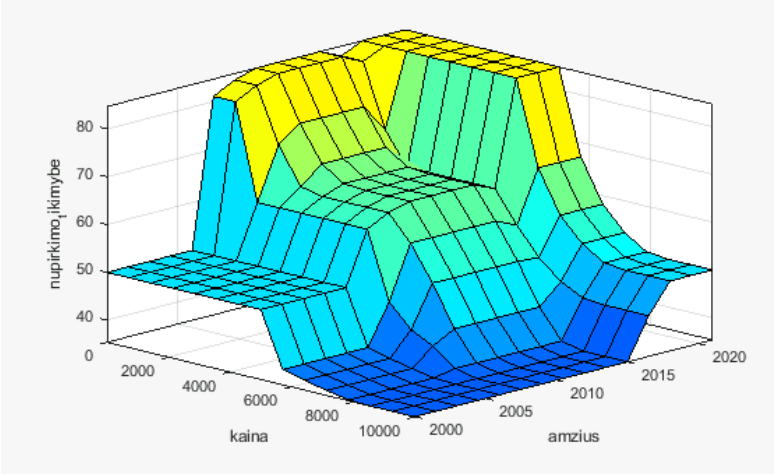
5 lentelė. *Sistemos taisyklės*

Kaina		Rida		Amžius	Nupirkimo tikimybė
aukšta	and	aukšta	and	senas	maža
aukšta	or	aukšta			maža
aukšta	and	vidutinė	and	vidutinis	maža
aukšta	and	not naujas			maža
maža	and	žema	and	naujas	maža
vidutinė	and	vidutinė	and	vidutinis	vidutinė
maža	and	žema	and	senas	vidutinė
aukšta	and	naujas			vidutinė
aukšta	and	not aukšta			vidutinė
vidutinė	and	žema	and	naujas	aukšta

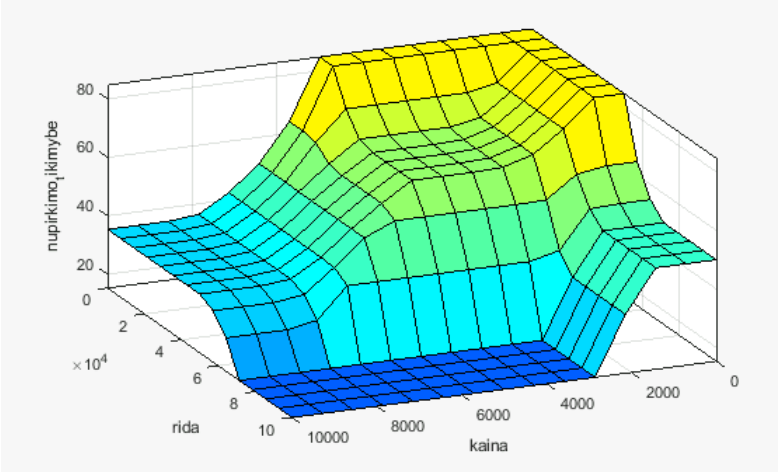
maža	and	vidutinis			aukšta
not aukšta	and	not aukšta	and	vidutinis	aukšta
vidutinė			and	naujas	aukšta

1.3. Sistemos patikrinimas per Matlab

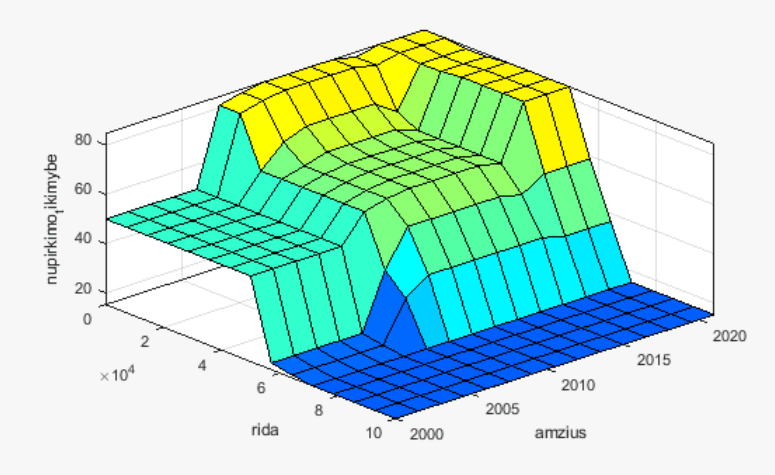
Sėkmingam realizavimui Python kalba, iš pradžių pasitelktas Matlab įrankis fuzzy miglotosios logikos projektavimui. Šis įrankis leis pasitikrinti taisyklių korektiškumą pavaizduojant dviejų parametrų ir rezultato parametro paviršių. Šie paviršiai pateikti žemiau.



5 pav. *Kainos ir amžiaus paviršius*



6 pav. *Kainos ir ridos paviršius*



7 pav. *Ridos ir amžiaus paviršius*

2. Python sistemos tikrinimas

Realizuotos sistemos patikrinimui bus apskaičiuojami trys scenarijai testavimui, pateikti 6 lentelėje. Šios vertės parinktos pavaizduoti miglotos logikos taisyklių persidengimui, ir pavaizduoti rezultatus, kai aktyvuota tik viena implikacija.

6 lentelė. *Testinės reikšmės*

Scenarijus	Kaina, €	Rida, km	Amžius, m
1	1990	62000	2008
2	9096	80120	2000
3	1506	19880	2013

2.1. Įvesties teiginių aktyvavimas

Testavimui parinktos reikšmės, kurios leistų išbandyti agregavimą, kai aktyvuojamos kelios taisyklės, bei kai aktyvuojama tik viena ir pasižiūrėti skirtumą tarp skirtingų defuzifikacijos metodų.

2.1.1. Kainos įverčiai

7 lentelėje pateikti kainos aktyvavimo įverčiai. Matome, jog pirmajame scenarijuje gana lygiai aktyvuoti mažos ir vidutinės kainos įverčiai, antrajame parinka tik vieną kainos kategoriją atitinkančią vertę, o trečiame scenarijuje parinktas tarpinis variantas.

7 lentelė. *Kainos aktyvacijos*

Scenarijus	Maža	Vidutinė	Didelė
1	0.51	0.495	0
2	0	0	1
3	0.994	0.253	0

2.1.2. Ridos įverčiai

8 lentelėje pateikti ridos aktyvavimo įverčiai.

8 lentelė. *Ridos aktyvacijos*

Scenarijus	Maža	Vidutinė	Didelė
1	0	0.4	0.6
2	0	0	1
3	0.34	0.66	0

2.1.3. Amžiaus įverčiai

9 lentelėje pateikti amžiaus aktyvavimo įverčiai.

9 lentelė. *Amžiaus aktyvacijos*

Scenarijus	Sena	Vidutinė	Nauja
1	0.67	0.33	0
2	1	0	0
3	0	1	0

2.2. Implikacija

Gautos kiekvieno scenarijaus automobilio nupirkimo tikimybės pateiktos 10 lentelėje.

10 lentelė. *Rezultatų implikacijos*

Scenarijus	Žema	Vidutinė	Aukšta
1	0.6	0.51	0.33
2	1	0	0
3	0	0.253	1

2.3. Agregacija ir defuzifikacija

Defuzifikacijai pasirinkta realizuoti du metodus – centroido ir maksimumų vidurkio (MOM).

Dėl paprastumo centroido radimo būdas programiškai realizuotas naudojant skaitinį integralo sprendimo būdą – figūra suskaidoma į daugybę plonų pasuktų trapecijų, kurių kiekvienos masės centras randamas pagal (1) formulę.

$$c_x = \frac{\Delta x(2y_1 + y_2)}{3(y_1 + y_2)} \tag{1}$$

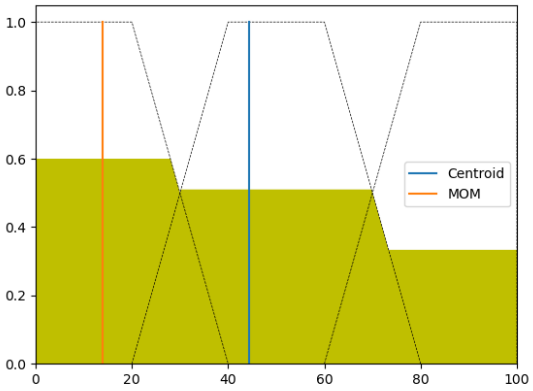
Čia: Δx – pasuktos trapecijos aukštis, y_1 – y vertė mažosios atkarpos pradžioje, y_2 – y vertė mažosios atkarpos gale, c_x – plonos atkarpos centroido vertė (prie jos pridėjus x pradžios vertę, tai leis apskaičiuoti bendros agreguotos figūros masės centro reikšmę x ašyje.

Vidurkių maksimumo metodą programiškai dar paprasčiau realizuoti – naudojant masyvų filtravimą kaukėmis grąžinami tik maksimalią reikšmę turintys elementai, ir randamas jų vidurkis. Tai matoma programinio kodo skyriuje, funkcijoje „mom“.

Kiekvieno scenarijaus suagreguoti išvesties įverčiai pateikti paveiksliukuose žemiau. Naudojant vertikalias linijas pažymėti rezultatai po defuzifikacijos, žaliu fonu – suagreguotos išvesties reikšmės.

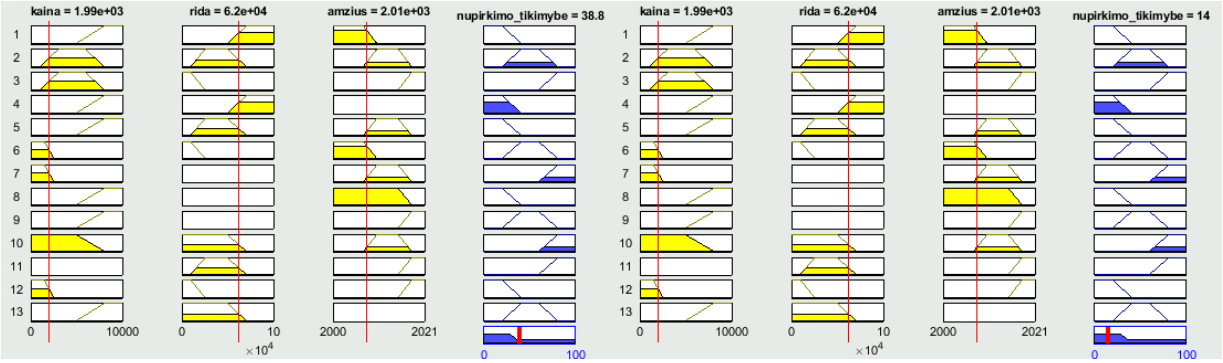
Pirmojo scenarijaus rezultatai (automobilio nupirkimo tikimybė):

- 44.4% (centroidas);
- 14% (MOM).



8 pav. *Pirmojo scenarijaus agregacijos ir defuzifikacijos diagrama*

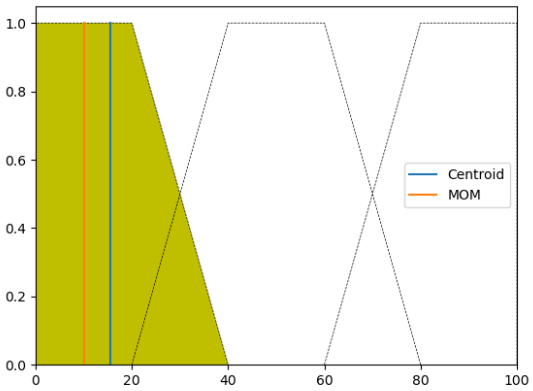
Patikrinimui 9 pav. pateiki rezultatai gaunami naudojant Matlab fuzzy įrankį.



9 pav. *Pirmojo scenarijaus rezultatai iš Matlab*

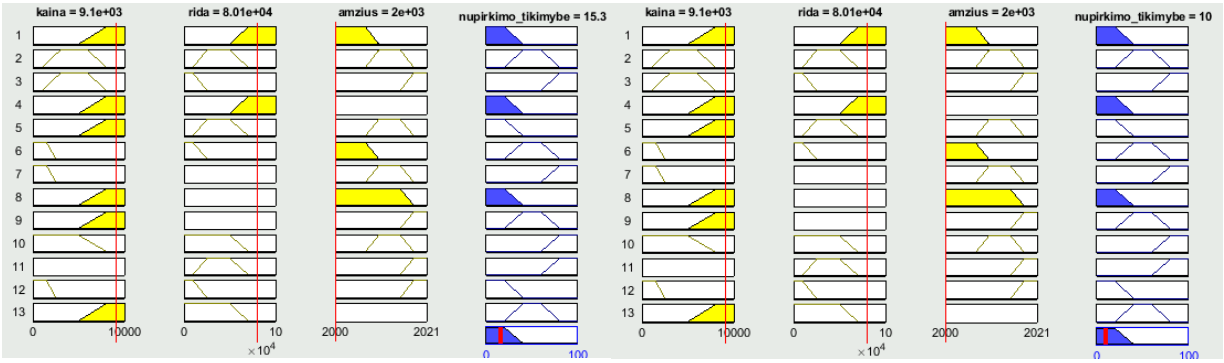
Antrojo scenarijaus rezultatai (automobilio nupirkimo tikimybė):

- 15.6% (centroidas);
- 10% (MOM).



10 pav. *Antrojo scenarijaus agregacijos ir defuzifikacijos diagrama*

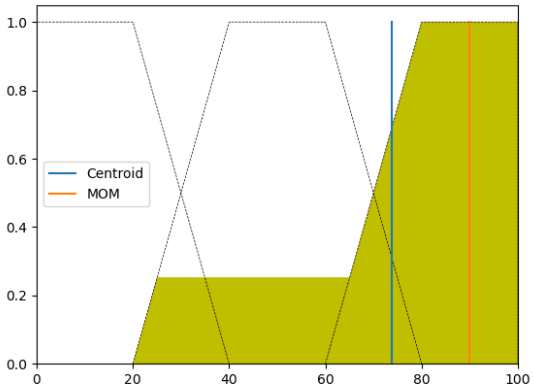
Patikrinimui 11 pav. pateiki rezultatai gaunami naudojant Matlab fuzzy įrankį.



11 pav. *Antrojo scenarijaus rezultatai iš Matlab*

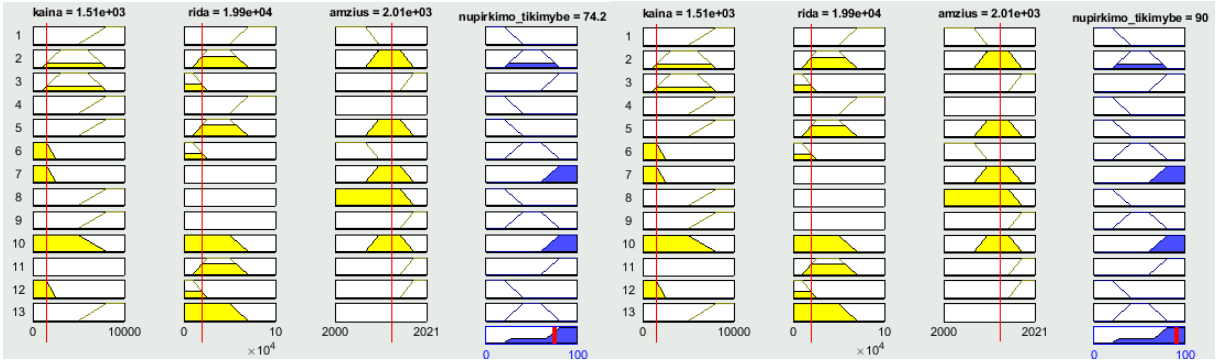
Trečiojo scenarijaus rezultatai (automobilio nupirkimo tikimybė):

- 73.9% (centroidas);
- 90% (MOM).



12 pav. *Trečiojo scenarijaus agregacijos ir defuzifikacijos diagrama*

Patikrinimui 13 pav. pateiki rezultatai gaunami naudojant Matlab fuzzy įrankį.



13 pav. *Trečiojo scenarijaus rezultatai iš Matlab*

3. Išvados

Sėkmingai pavyko realizuoti automobilių pirkimo tikimybės modelį. Tai patikrinta naudojant Matlab kaip pagalbinį įrankį.

Modelio patikrinimas naudojant Matlab leido toliau sėkmingai sukurti modelį naudojant Python programavimo kalbą išvengiant grubių modelio klaidų.

Rastas skirtumas tarp centroido ir vidurkių maksimumo metodų – pirmasis atsižvelgia į visas agregacijos reikšmes, o MOM metodas labiau tinka, kai pagrinde aktyvuotas vienas iš rezultatų įverčių.

4. Programinis kodas

```
import numpy as np
from matplotlib import pyplot as plt

STEP = 0.01

LOW_PROB = [0, 0, 20, 40]
MID_PROB = [20, 40, 60, 80]
HIGH_PROB = [60, 80, 100, 100]


def trap_y(x, a, b, c, d):
    if a <= x < b:
        return (x - a) / (b - a)
    elif b <= x < c:
        return 1
    elif c <= x < d:
        return (d - x) / (d - c)
    else:
        return 0


def trap_x(y, a, b, c, d):
    return a + (b - a) * y, d - (d - c) * y


def price_activation(x):
    points = [
        [0, 0, 1500, 2500],  # Cheap
        [1000, 3000, 6000, 8000],  # Average
        [5000, 8000, 10000, 10000]  # Expensive
    ]
    return [trap_y(x, *p) for p in points]


def mileage_activation(x):
    points = [
        [0, 0, 10000, 25000],  # Low
        [10000, 25000, 50000, 70000],  # Medium
        [50000, 70000, 100001, 100001]  # High
    ]
    return [trap_y(x, *p) for p in points]


def age_activation(x):
    points = [
        [2000, 2000, 2007, 2010],  # Old
        [2007, 2010, 2015, 2018],  # Average
        [2015, 2018, 2022, 2022]  # New
    ]
    return [trap_y(x, *p) for p in points]


def low_prob_rules(pw, mw, aw):
    weights = [min(pw[2], mw[2], aw[0]),
               max(pw[2], mw[2]),
               min(pw[2], mw[1], aw[1]),
               min(pw[2], 1 - aw[1]),
               min(pw[0], mw[0], aw[2])]

    return max(weights)


def mid_prob_rules(pw, mw, aw):
    weights = [min(pw[1], mw[1], aw[1]),
               min(pw[0], mw[2], aw[0]),
               min(pw[2], aw[2]),
               min(pw[2], 1 - mw[2])]

    return max(weights)


def high_prob_rules(pw, mw, aw):
    weights = [min(pw[1], mw[0], aw[2]),
```

```
                min(pw[0], aw[1]),
                min(1 - pw[2], 1 - mw[2], aw[1]),
                min(mw[1], aw[2])]

    return max(weights)


def result_implication(price, mileage, age):
    pw = price_activation(price)
    mw = mileage_activation(mileage)
    aw = age_activation(age)

    low = low_prob_rules(pw, mw, aw)
    med = mid_prob_rules(pw, mw, aw)
    high = high_prob_rules(pw, mw, aw)

    return low, med, high


def get_aggregate(low, med, high):
    x = np.arange(0, 100, STEP)
    aggregate = np.zeros_like(x)

    xi = 0.0
    for i in range(len(aggregate)):
        low_step = min(low, trap_y(xi, *LOW_PROB))
        med_step = min(med, trap_y(xi, *MID_PROB))
        high_step = min(high, trap_y(xi, *HIGH_PROB))
        xi += STEP

        aggregate[i] = max(low_step, med_step, high_step)

    return x, aggregate


def centroid(low, med, high):
    x, aggregate = get_aggregate(low, med, high)

    sum_moment_area = 0.0
    sum_area = 0.0

    x = np.arange(0, 100, STEP)
    for i in range(1, len(aggregate)):
        y1 = aggregate[i - 1]
        y2 = aggregate[i]

        if not (y1 == y2 == 0.0):
            moment = STEP * (2 * y1 + y2) / (3 * (y1 + y2)) + x[i - 1]
            area = STEP * (y1 + y2) / 2

            sum_moment_area += moment * area
            sum_area += area

    return sum_moment_area / sum_area


def mom(low, med, high):
    x, aggregate = get_aggregate(low, med, high)
    return np.mean(x[aggregate == aggregate.max()])


def main():
    price = 1055
    mileage = 33500
    age = 2017

    low, med, high = result_implication(price, mileage, age)
    centroid_res = centroid(low, med, high)
    mom_res = mom(low, med, high)

    print('Purchase probability (centroid) = {:.1f}'.format(centroid_res))
    print('Purchase probability (MOM) = {:.1f}'.format(mom_res))

    # Outline
    plt.plot(LOW_PROB, [0, 1, 1, 0], 'k--', linewidth=0.5)
    plt.plot(MID_PROB, [0, 1, 1, 0], 'k--', linewidth=0.5)
```

```
    plt.plot(HIGH_PROB, [0, 1, 1, 0], 'k--', linewidth=0.5)

    # Activated purchase probabilities
    plt.fill([0, *trap_x(low, *LOW_PROB), 40], [0, low, low, 0], 'y')
    plt.fill([20, *trap_x(med, *MID_PROB), 80], [0, med, med, 0], 'y')
    plt.fill([60, *trap_x(high, *HIGH_PROB), 100], [0, high, high, 0], 'y')

    # Result with centroid defuzzification
    cen_ax, = plt.plot([centroid_res, centroid_res], [0, 1])
    # Result with MOM defuzzification
    mom_ax, = plt.plot([mom_res, mom_res], [0, 1])

    plt.xlim(0, 100)
    plt.ylim(0, 1.05)
    plt.legend([cen_ax, mom_ax], ['Centroid', 'MOM'])
    plt.show()


if __name__ == '__main__':
    main()
```